Pepe Gallardo – Universidad de Málaga

Tema 9. Árboles

9.1 Árboles binarios

✓ Tipo para árboles binarios homogéneos con datos en nodos:

$$\textbf{data}\ \ \acute{A}rbolB\ a\ =\ \ VacíoB \mid NodoB\ \overbrace{(\acute{A}rbolB\ a)}^{hijo\ izq}\ \ \overbrace{a}^{dato\ en\ nodo}\ \ \overbrace{(\acute{A}rbolB\ a)}^{hijo\ der}$$
$$\textbf{deriving}\ Show$$

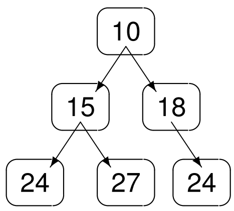
✓ $VacíoB$ es un árbol binario vacío

✓ Las tres componentes de $NodoB$ son

- ⋄ subárbol izquierdo
- ⋄ dato en nodo
- ⋄ subárbol derecho

✓ Si falta un subárbol, se usa $VacíoB$.

Ejemplo:



$$
\begin{aligned}
&a\ ::\ \acute{A}rbolB\ Integer\\
&a\ =\ NodoB\ aI\ 10\ aD\\
&\quad \textbf{where}\\
&\qquad aI\ \ = NodoB\ aII\ 15\ aID\\
&\qquad aD\ \ = NodoB\ VacíoB\ 18\ aDD\\
&\qquad aII\ = hojaB\ 24\\
&\qquad aID\ = hojaB\ 27\\
&\qquad aDD = hojaB\ 24
\end{aligned}
$$

$$
\begin{aligned}
&hojaB\ \ \ ::\ a \rightarrow \acute{A}rbolB\ a\\
&hojaB\ x\ =\ NodoB\ VacíoB\ x\ VacíoB
\end{aligned}
$$

Árboles binarios (2)

$$\textbf{data}\ \textit{ÁrbolB}\ a = \textit{VacíoB} \mid \textit{NodoB}\ \overbrace{(\textit{ÁrbolB}\ a)}^{\textit{hijo izq}}\ \overbrace{a}^{\textit{dato en nodo}}\ \overbrace{(\textit{ÁrbolB}\ a)}^{\textit{hijo der}}\quad \textbf{deriving}\ \textit{Show}$$

✓ Algunas funciones

$$\begin{array}{lcl}
\textit{raízB} & :: & \textit{ÁrbolB}\ a \to a \\
\textit{raízB}\ \textit{VacíoB} & = & \textit{error}\ \text{"raíz de árbol vacío"} \\
\textit{raízB}\ (\textit{NodoB}\ _\ x\ _) & = & x
\end{array}$$

$$\begin{array}{lcl}
\textit{tamañoB} & :: & \textit{ÁrbolB}\ a \to \textit{Integer} \\
\textit{tamañoB}\ \textit{VacíoB} & = & 0 \\
\textit{tamañoB}\ (\textit{NodoB}\ i\ _\ d) & = & 1 + \textit{tamañoB}\ i + \textit{tamañoB}\ d
\end{array}$$

$$\begin{array}{lcl}
\textit{profundidadB} & :: & \textit{ÁrbolB}\ a \to \textit{Integer} \\
\textit{profundidadB}\ \textit{VacíoB} & = & 0 \\
\textit{profundidadB}\ (\textit{NodoB}\ i\ _\ d) & = & 1 + \textit{max}\ (\textit{profundidadB}\ i)\ (\textit{profundidadB}\ d)
\end{array}$$

✓ Recorridos de un árbol binario:

$$\begin{array}{lcl}
\textit{enOrdenB} & :: & \textit{ÁrbolB}\ a \to [a] \\
\textit{enOrdenB}\ \textit{VacíoB} & = & [\,] \\
\textit{enOrdenB}\ (\textit{NodoB}\ i\ r\ d) & = & \textit{enOrdenB}\ i +\!\!+ [r] +\!\!+ \textit{enOrdenB}\ d
\end{array}$$

$$\begin{array}{lcl}
\textit{preOrdenB} & :: & \textit{ÁrbolB}\ a \to [a] \\
\textit{preOrdenB}\ \textit{VacíoB} & = & [\,] \\
\textit{preOrdenB}\ (\textit{NodoB}\ i\ r\ d) & = & [r] +\!\!+ \textit{preOrdenB}\ i +\!\!+ \textit{preOrdenB}\ d
\end{array}$$

$$\begin{array}{lcl}
\textit{postOrdenB} & :: & \textit{ÁrbolB}\ a \to [a] \\
\textit{postOrdenB}\ \textit{VacíoB} & = & [\,] \\
\textit{postOrdenB}\ (\textit{NodoB}\ i\ r\ d) & = & \textit{postOrdenB}\ i +\!\!+ \textit{postOrdenB}\ d +\!\!+ [r]
\end{array}$$

fmap para árboles binarios

✓ La función map solo está definidad para listas

$$\begin{array}{lcl} map & :: & (a \to b) \to [a] \to [b] \\ map\ f\ [\,] & = & [\,] \\ map\ f\ (x : xs) & = & f\ x\ :\ map\ f\ xs \end{array}$$

✓ La clase $Functor$ define una función $fmap$ para estructuras arbitrarias

$$\begin{array}{l} \mathbf{class}\ Functor\ t\ \mathbf{where} \\ \quad fmap\ ::\ (a \to b) \to t\ a \to t\ b \end{array}$$

✓ Las listas son una instancia predefinida:

$$\begin{array}{l} \mathbf{instance}\ Functor\ [\,]\ \mathbf{where} \\ \quad fmap\ =\ map \end{array}$$

✓ Es posible usar tanto map como $fmap$ con listas.

✓ La función $fmap$ también tiene sentido para árboles binarios:

$$\begin{array}{l} \mathbf{instance}\ Functor\ \acute{A}rbolB\ \mathbf{where} \\ \quad \begin{array}{lcl} fmap\ f\ Vac\acute{\imath}oB & = & Vac\acute{\imath}oB \\ fmap\ f\ (NodoB\ i\ r\ d) & = & NodoB\ (fmap\ f\ i)\ (f\ r)\ (fmap\ f\ d) \end{array} \end{array}$$

✓ Ejemplo de uso:

$$\begin{array}{lcl} empareja & :: & Functor\ t\ \Rightarrow t\ a \to t\ (a, a) \\ empareja & = & fmap\ (\lambda\ x \to (x, x)) \end{array}$$

? $empareja$ [1..3]
[(1, 1), (2, 2), (3, 3)] :: [($Integer$, $Integer$)]

? $empareja\ (NodoB\ Vac\acute{\imath}oB$ 10 $Vac\acute{\imath}oB)$
$NodoB\ Vac\acute{\imath}oB$ (10, 10) $Vac\acute{\imath}oB$ :: $\acute{A}rbolB\ (Integer, Integer)$

Plegado de árboles binarios

✓ Muchas funciones sobre el tipo $\acute{A}rbolB$ siguen el mismo esquema:

$$
\begin{array}{lcl}
sum\acute{A}rbolB & :: & \acute{A}rbolB\ Integer \rightarrow Integer \\
sum\acute{A}rbolB\ Vac\acute{\imath}oB & = & 0 \\
sum\acute{A}rbolB\ (NodoB\ i\ r\ d) & = & sumar\ (sum\acute{A}rbolB\ i)\ r\ (sum\acute{A}rbolB\ d) \\
\quad \textbf{where} & & \\
\qquad sumar\ x\ y\ z\ =\ x + y + z & &
\end{array}
$$

$$
\begin{array}{lcl}
enOrdenB & :: & \acute{A}rbolB\ t \rightarrow [t] \\
enOrdenB\ Vac\acute{\imath}oB & = & [\,] \\
enOrdenB\ (NodoB\ i\ r\ d) & = & concatenar\ (enOrdenB\ i)\ r\ (enOrdenB\ d) \\
\quad \textbf{where} & & \\
\qquad concatenar\ x\ y\ z\ =\ x +\!\!+ [y] +\!\!+ z & &
\end{array}
$$

✓ El esquema común y la función de orden superior:

$$
\begin{array}{lcl}
fun & :: & \acute{A}rbolB\ a \rightarrow b \\
fun\ Vac\acute{\imath}oB & = & \boxed{z} \\
fun\ (NodoB\ i\ r\ d) & = & \boxed{f}\ (fun\ i)\ r\ (fun\ d)
\end{array}
$$

$$
\begin{array}{lcl}
fold\acute{A}rbolB & :: & (b \rightarrow a \rightarrow b \rightarrow b) \rightarrow b \rightarrow (\acute{A}rbolB\ a \rightarrow b) \\
fold\acute{A}rbolB\ f\ z & = & fun \\
\quad \textbf{where} & & \\
\qquad fun\ Vac\acute{\imath}oB & = & z \\
\qquad fun\ (NodoB\ i\ r\ d) & = & f\ (fun\ i)\ r\ (fun\ d)
\end{array}
$$

✓ O equivalentemente (ya que $fold\acute{A}rbolB\ f\ z\ =\ fun$)

$$
\begin{array}{lcl}
fold\acute{A}rbolB & :: & (b \rightarrow a \rightarrow b \rightarrow b) \rightarrow b \rightarrow \acute{A}rbolB\ a \rightarrow b \\
fold\acute{A}rbolB\ f\ z\ Vac\acute{\imath}oB & = & z \\
fold\acute{A}rbolB\ f\ z\ (NodoB\ i\ r\ d) & = & f\ (fold\acute{A}rbolB\ f\ z\ i)\ r\ (fold\acute{A}rbolB\ f\ z\ d)
\end{array}
$$

✓ Las funciones originales como concreción de $fold\acute{A}rbolB$:

$$
\begin{array}{lcl}
sum\acute{A}rbolB & :: & \acute{A}rbolB\ Integer \rightarrow Integer \\
sum\acute{A}rbolB & = & fold\acute{A}rbolB\ (\lambda\ x\ y\ z \rightarrow x + y + z)\ 0
\end{array}
$$

$$
\begin{array}{lcl}
enOrdenB & :: & \acute{A}rbolB\ t \rightarrow [t] \\
enOrdenB & = & fold\acute{A}rbolB\ (\lambda\ x\ y\ z \rightarrow x +\!\!+ [y] +\!\!+ z)\ [\,]
\end{array}
$$

Plegado de árboles binarios (2)

$$
\begin{array}{lcl}
fold\acute{A}rbolB & :: & (b \to a \to b \to b) \to b \to (\acute{A}rbolB\ a \to b) \\
fold\acute{A}rbolB\ f\ z & = & fun \\
\quad \mathbf{where} & & \\
\qquad fun\ Vac\acute{\imath}oB & = & z \\
\qquad fun\ (NodoB\ i\ r\ d) & = & f\ (fun\ i)\ r\ (fun\ d)
\end{array}
$$

- Para definir una función usando $fold\acute{A}rbolB$:
  - Proporcionar el resultado para el árbol vacío (z) .
  - Proporcionar función (f) que calcule el resultado a partir del resultado para el subárbol izquierdo, la raíz y el resultado para el subárbol derecho.

Ejemplos

- tamaño de un árbol (número de elementos almacenados)

$$
\begin{array}{lcl}
tama\tilde{n}oB & :: & \acute{A}rbolB\ a \to Integer \\
tama\tilde{n}oB & = & fold\acute{A}rbolB\ (\lambda\ ti\ r\ td \to 1\ +\ ti\ +\ td)\ 0
\end{array}
$$

- profundidad de un árbol

$$
\begin{array}{lcl}
profundidadB & :: & \acute{A}rbolB\ a \to Integer \\
profundidadB & = & fold\acute{A}rbolB\ (\lambda\ pi\ r\ pd \to 1\ +\ max\ pi\ pd)\ 0
\end{array}
$$

- recorrido en pre orden

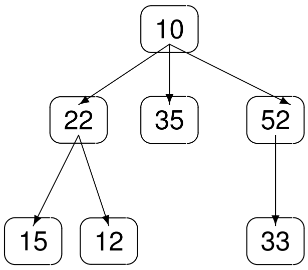
$$
\begin{array}{lcl}
preOrdenB & :: & \acute{A}rbolB\ a \to [a] \\
preOrdenB & = & fold\acute{A}rbolB\ (\lambda\ pi\ r\ pd \to [r] +\!\!+\ pi\ +\!\!+\ pd)\ []
\end{array}
$$

9.2 Árboles generales

✓ Un árbol es una estructura no lineal acíclica

✓ Un árbol es una colección de valores $\{v_1,\ v_2,\ \ldots\ v_n\}$ tales que

- ◇ Si $n = 0$, el árbol se dice *vacío*
- ◇ En otro caso, existe un valor destacado (la *raíz*) y los demás elementos forman parte de colecciones disjuntas que a su vez son árboles (*subárboles* del raíz)



$$\textbf{data}\ \textit{Árbol}\ a = \textit{Vacío} \mid \textit{Nodo}\ \underbrace{a}_{\textit{raíz}}\ \underbrace{[\textit{Árbol}\ a]}_{\textit{hijos}}\ \textbf{deriving}\ \textit{Show}$$

$$\begin{array}{l}
a1 :: \textit{Árbol Integer} \\
a1 = \textit{Nodo}\ 10\ [a11,\ a12,\ a13] \\
\quad \textbf{where} \\
\qquad a11 = \textit{Nodo}\ 22\ [\textit{hoja}\ 15,\ \textit{hoja}\ 12] \\
\qquad a12 = \textit{hoja}\ 35 \\
\qquad a13 = \textit{Nodo}\ 52\ [\textit{hoja}\ 33]
\end{array}$$

$$\begin{array}{lcl}
\textit{hoja} & :: & a \rightarrow \textit{Árbol}\ a \\
\textit{hoja}\ x & = & \textit{Nodo}\ x\ [\,]
\end{array}$$

$$\begin{array}{lcl}
\textit{raíz} & :: & \textit{Árbol}\ a \rightarrow a \\
\textit{raíz Vacío} & = & \textit{error}\ \text{"raíz de árbol vacío"} \\
\textit{raíz}\ (\textit{Nodo}\ x\ _\,) & = & x
\end{array}$$

$$\begin{array}{lcl}
\textit{subárboles} & :: & \textit{Árbol}\ a \rightarrow [\textit{Árbol}\ a] \\
\textit{subárboles Vacío} & = & \textit{error}\ \text{"subárboles de árbol vacío"} \\
\textit{subárboles}\ (\textit{Nodo}\ _\ xs) & = & xs
\end{array}$$

Árboles generales (2)

$$\textbf{data}\ \textit{Árbol}\ a = \textit{Vacío} \mid \textit{Nodo}\ \underbrace{a}_{\textit{raíz}}\ \underbrace{[\textit{Árbol}\ a]}_{\textit{hijos}}\ \textbf{deriving}\ \textit{Show}$$

✓ Algunas funciones

$$\begin{array}{lcl}
\textit{tamaño} & :: & \textit{Árbol}\ a \to \textit{Integer} \\
\textit{tamaño Vacío} & = & 0 \\
\textit{tamaño}\ (\textit{Nodo}\ _\ xs) & = & 1 + \textit{sum}\ (\textit{map tamaño}\ xs)
\end{array}$$

$$\begin{array}{lcl}
\textit{profundidad} & :: & \textit{Árbol}\ a \to \textit{Integer} \\
\textit{profundidad Vacío} & = & 0 \\
\textit{profundidad}\ (\textit{Nodo}\ _\ [\,]) & = & 1 \\
\textit{profundidad}\ (\textit{Nodo}\ _\ xs) & = & 1 + \textit{maximum}\ (\textit{map profundidad}\ xs)
\end{array}$$

$$\begin{array}{lcl}
\textit{pertenece} & :: & \textit{Eq}\ a \Rightarrow a \to \textit{Árbol}\ a \to \textit{Bool} \\
\textit{pertenece}\ _\ \textit{Vacío} & = & \textit{False} \\
\textit{pertenece}\ x\ (\textit{Nodo}\ r\ xs) & = & x == r\ \mid\mid\ \textit{or}\ (\textit{map}\ (\textit{pertence}\ x)\ xs)
\end{array}$$

✓ Recorridos de un árbol general:

$$\begin{array}{lcl}
\textit{preOrden} & :: & \textit{Árbol}\ a \to [a] \\
\textit{preOrden Vacío} & = & [\,] \\
\textit{preOrden}\ (\textit{Nodo}\ r\ xs) & = & [r] +\!\!+\ \textit{concat}\ (\textit{map preOrden}\ xs)
\end{array}$$

$$\begin{array}{lcl}
\textit{postOrden} & :: & \textit{Árbol}\ a \to [a] \\
\textit{postOrden Vacío} & = & [\,] \\
\textit{postOrden}\ (\textit{Nodo}\ r\ xs) & = & \textit{concat}\ (\textit{map postOrden}\ xs) +\!\!+ [r]
\end{array}$$

fmap para árboles generales

✓ La función fmap también tiene sentido para un árbol general

$$\begin{array}{l} \textbf{instance}\ \mathit{Functor}\ \mathit{Árbol}\ \textbf{where} \\ \quad \begin{array}{lcl} \mathit{fmap}\ f\ \mathit{Vacío} & = & \mathit{Vacío} \\ \mathit{fmap}\ f\ (\mathit{Nodo}\ r\ \mathit{xs}) & = & \mathit{Nodo}\ (f\ r)\ [\ \mathit{fmap}\ f\ x \mid x \leftarrow \mathit{xs}\ ] \end{array} \end{array}$$

o usando map

$$\begin{array}{l} \textbf{instance}\ \mathit{Functor}\ \mathit{Árbol}\ \textbf{where} \\ \quad \begin{array}{lcl} \mathit{fmap}\ f\ \mathit{Vacío} & = & \mathit{Vacío} \\ \mathit{fmap}\ f\ (\mathit{Nodo}\ r\ \mathit{xs}) & = & \mathit{Nodo}\ (f\ r)\ (\mathit{map}\ (\mathit{fmap}\ f)\ \mathit{xs}) \end{array} \end{array}$$

✓ Ahora la función $\mathit{empareja}$ puede usarse con árboles generales

$$\begin{array}{lcl} \mathit{empareja} & :: & \mathit{Functor}\ t\ \Rightarrow t\ a \rightarrow t\ (a, a) \\ \mathit{empareja} & = & \mathit{fmap}\ (\lambda\ x \rightarrow (x, x)) \end{array}$$

? $\mathit{empareja}\ (\mathit{Nodo}\ 10\ [\mathit{Nodo}\ 20\ [\,]])$
$\mathit{Nodo}\ (10, 10)\ [\mathit{Nodo}\ (20, 20)\ [\,]]\ \ ::\ \mathit{Árbol}\ (\mathit{Integer}, \mathit{Integer})$

? $\mathit{fmap}\ (+1)\ (\mathit{Nodo}\ 10\ [\mathit{Nodo}\ 20\ [\,]])$
$\mathit{Nodo}\ 11\ [\mathit{Nodo}\ 21\ [\,]]\ \ ::\ \mathit{Árbol}\ \mathit{Integer}$

Plegado de árboles generales

✓ Muchas funciones siguen el mismo esquema

$$
\begin{array}{lcl}
sum\acute{A}rbol & :: & \acute{A}rbol\ Integer \rightarrow Integer \\
sum\acute{A}rbol\ Vac\acute{\imath}o & = & 0 \\
sum\acute{A}rbol\ (Nodo\ r\ xs) & = & sumar\ r\ (map\ sum\acute{A}rbol\ xs) \\
\quad \mathbf{where} & & \\
\qquad sumar\ y\ ys = y + sum\ ys & & \\
preOrden & :: & \acute{A}rbolB\ t \rightarrow [t] \\
preOrden\ Vac\acute{\imath}o & = & [\,] \\
preOrden\ (Nodo\ r\ xs) & = & unir\ r\ (map\ preOrden\ xs) \\
\quad \mathbf{where} & & \\
\qquad unir\ y\ ys = [y] \mathbin{+\!\!+} concat\ ys & &
\end{array}
$$

✓ El esquema común y la función de orden superior

$$
\begin{array}{lcl}
fun & :: & \acute{A}rbol\ a \rightarrow b \\
fun\ Vac\acute{\imath}o & = & \boxed{z} \\
fun\ (Nodo\ r\ xs) & = & \boxed{f}\ r\ (map\ fun\ xs)
\end{array}
$$

$$
\begin{array}{lcl}
fold\acute{A}rbol & :: & (a \rightarrow [b] \rightarrow b) \rightarrow b \rightarrow (\acute{A}rbol\ a \rightarrow b) \\
fold\acute{A}rbol\ f\ z & = & fun \\
\quad \mathbf{where} & & \\
\qquad fun\ Vac\acute{\imath}o & = & z \\
\qquad fun\ (Nodo\ r\ xs) & = & f\ r\ (map\ fun\ xs)
\end{array}
$$

✓ Las funciones originales como concreciones de $fold\acute{A}rbol$

$$
\begin{array}{lcl}
sum\acute{A}rbol & :: & \acute{A}rbol\ Integer \rightarrow Integer \\
sum\acute{A}rbol & = & fold\acute{A}rbol\ (\lambda\ y\ ys \rightarrow y + sum\ ys)\ 0
\end{array}
$$

$$
\begin{array}{lcl}
preOrden & :: & \acute{A}rbolB\ t \rightarrow [t] \\
preOrden & = & fold\acute{A}rbol\ (\lambda\ y\ ys \rightarrow [y] \mathbin{+\!\!+} concat\ ys)\ [\,]
\end{array}
$$

Plegado de árboles generales (2)

$$
\begin{array}{lcl}
\mathit{fold\acute{A}rbol} & :: & (a \to [b] \to b) \to b \to (\mathit{\acute{A}rbol}\ a \to b) \\
\mathit{fold\acute{A}rbol}\ f\ z & = & \mathit{fun} \\
\quad \textbf{where} & & \\
\qquad \mathit{fun}\ \mathit{Vac\acute{\imath}o} & = & z \\
\qquad \mathit{fun}\ (\mathit{Nodo}\ r\ xs) & = & f\ r\ (\mathit{map}\ \mathit{fun}\ xs)
\end{array}
$$

✓ Para definir una función usando $\mathit{fold\acute{A}rbol}$

◇ Proporcionar el resultado para el árbol vacío (z) .

◇ Proporcionar función (f) que calcule el resultado a partir de la raíz y una lista con el resultado para cada subárbol.

Ejemplos

✓ tamaño de un árbol

$$
\begin{array}{lcl}
\mathit{tama\tilde{n}o} & :: & \mathit{\acute{A}rbol}\ a \to \mathit{Integer} \\
\mathit{tama\tilde{n}o} & = & \mathit{fold\acute{A}rbol}\ (\lambda\ r\ ts \to 1 + \mathit{sum}\ ts)\ 0
\end{array}
$$

✓ pertenencia de un dato a un árbol

$$
\begin{array}{lcl}
\mathit{pertenece} & :: & \mathit{Eq}\ a \Rightarrow a \to \mathit{\acute{A}rbol}\ a \to \mathit{Bool} \\
\mathit{pertenece}\ x & = & \mathit{fold\acute{A}rbol}\ (\lambda\ r\ ps \to x == r\ \|\ \mathit{or}\ ps)\ \mathit{False}
\end{array}
$$

✓ recorrido en post orden

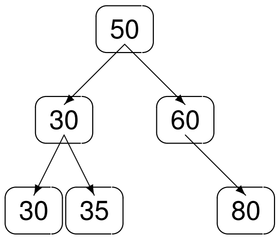
$$
\begin{array}{lcl}
\mathit{postOrden} & :: & \mathit{\acute{A}rbol}\ a \to [a] \\
\mathit{postOrden} & = & \mathit{fold\acute{A}rbol}\ (\lambda\ r\ ps \to \mathit{concat}\ ps +\!\!+ [r])\ [\,]
\end{array}
$$

9.3 Árboles de búsqueda

✓ Un árbol de búsqueda es un árbol binario tal que

- ◇ O bien es vacío
- ◇ O no es vacío y para cualquier nodo se cumple que:
  - ○ los elementos del subárbol izquierdo son menores o iguales al almacenado en el nodo
  - ○ y los elementos del subárbol derecho son estrictamente mayores al almacenado en el nodo

Ejemplo



✓ La siguiente función puede ser utilizada para comprobar si un árbol binario es de búsqueda:

$$
\begin{array}{lcl}
es\acute{A}rbolBB & :: & Ord\ a\ \Rightarrow \acute{A}rbolB\ a \rightarrow Bool \\
es\acute{A}rbolBB\ Vac\acute{i}oB & = & True \\
es\acute{A}rbolBB\ (NodoB\ i\ r\ d) & = & todos\acute{A}rbolB\ (<=\ r)\ i \\
 & & \&\&\ todos\acute{A}rbolB\ (>\ \ r)\ d \\
 & & \&\&\ es\acute{A}rbolBB\ i \\
 & & \&\&\ es\acute{A}rbolBB\ d
\end{array}
$$

$$
\begin{array}{lcl}
todos\acute{A}rbolB & :: & (a \rightarrow Bool) \rightarrow \acute{A}rbolB\ a \rightarrow Bool \\
todos\acute{A}rbolB\ p\ Vac\acute{i}oB & = & True \\
todos\acute{A}rbolB\ p\ (NodoB\ i\ r\ d) & = & p\ r\ \&\& \\
 & & todos\acute{A}rbolB\ p\ i\ \&\&\ todos\acute{A}rbolB\ p\ d
\end{array}
$$

Árboles de búsqueda (2)

✓ Pertenencia a un árbol de búsqueda

$$
\begin{array}{lcl}
\mathit{perteneceBB} & :: & \mathit{Ord}\ a\ \Rightarrow a \rightarrow \mathit{ÁrbolB}\ a \rightarrow \mathit{Bool} \\
\mathit{perteneceBB}\ x\ \mathit{VacíoB} & = & \mathit{False} \\
\mathit{perteneceBB}\ x\ (\mathit{NodoB}\ i\ r\ d) & & \\
\quad \mid\ x == r & = & \mathit{True} \\
\quad \mid\ x\ <\ r & = & \mathit{perteneceBB}\ x\ i \\
\quad \mid\ \mathit{otherwise} & = & \mathit{perteneceBB}\ x\ d
\end{array}
$$

✓ Inserción en un árbol de búsqueda

$$
\begin{array}{lcl}
\mathit{insertarBB} & :: & \mathit{Ord}\ a\ \Rightarrow a \rightarrow \mathit{ÁrbolB}\ a \rightarrow \mathit{ÁrbolB}\ a \\
\mathit{insertarBB}\ x\ \mathit{VacíoB} & = & \mathit{NodoB}\ \mathit{VacíoB}\ x\ \mathit{VacíoB} \\
\mathit{insertarBB}\ x\ (\mathit{NodoB}\ i\ r\ d) & & \\
\quad \mid\ x <= r & = & \mathit{NodoB}\ (\mathit{insertarBB}\ x\ i)\ r\ d \\
\quad \mid\ \mathit{otherwise} & = & \mathit{NodoB}\ i\ r\ (\mathit{insertarBB}\ x\ d)
\end{array}
$$

✓ Construcción de un árbol de búsqueda a partir de una lista

$$
\begin{array}{lcl}
\mathit{listaAÁrbolBB} & :: & \mathit{Ord}\ a\ \Rightarrow [a] \rightarrow \mathit{ÁrbolB}\ a \\
\mathit{listaAÁrbolBB} & = & \mathit{foldr}\ \mathit{insertarBB}\ \mathit{VacíoB}
\end{array}
$$

✓ El recorrido en orden genera una lista ordenada (*tree sort*)

$$
\begin{array}{lcl}
\mathit{treeSort} & :: & \mathit{Ord}\ a\ \Rightarrow [a] \rightarrow [a] \\
\mathit{treeSort} & = & \mathit{enOrdenB}\ .\ \mathit{listaAÁrbolBB}
\end{array}
$$

? $\mathit{treeSort}\ [4,7,1,2,9]$
$[1,2,4,7,9]\ ::\ [\mathit{Integer}]$

Objetivos del tema

El alumno debe:

- ✓ Conocer las definiciones de tipo para representar árboles en Haskell
- ✓ Saber definir funciones sobre árboles
- ✓ Conocer la clase $Functor$ y saber realizar instancias de esta clase
- ✓ Conocer las funciones de plegado para árboles
- ✓ Saber definir funciones como concreciones de las funciones de plegado
- ✓ Conocer la implementación de los árboles de búsqueda en Haskell